METHOD, SYSTEM AND COMPUTER PROGRAM PRODUCT FOR HIERARCHICAL LOAD BALANCING

Background of the Invention

Field of the Invention

5   The present invention relates generally to an improved system, method, and

computer program product for providing load balancing. A more particular aspect

is related to a system, method, and computer program product for hierarchical load

balancing, wherein frequently requested objects (e.g., web objects) are handled by a

front end cache and objects not in the cache are routed to back-end servers. The

10   popular requests are serviced directly from the cache and the remaining requests are

"URL hashed" to determine the destination server in a known manner.

Background of the Invention

The traffic on the World Wide Web ("The Web") is increasing exponentially.

The Web is used for a multitude of tasks, including information gathering,

15   electronic commerce, communication, and news dissemination. As a result of this

high traffic volume, systems have been developed to distribute Web traffic to

minimize waiting time for users.

Many of today's web sites are hosted on server farms, where a number of

servers are grouped together to serve web requests as they arrive. To avoid

20   overloading individual servers within the farm, load balancing techniques balance

the load across servers so that the best total throughput of the farm and smallest

response delay for the user is achieved.   Typically, a "server switch" performs

request distribution for the server farms, utilizing various techniques for determining

the destination server to handle the request. One such technique, called Server Load

5   Balancing (SLB), monitors at short, periodic intervals the load of the servers in the

farm and distributes incoming requests to the least loaded server.

Content Based Routing (CBR) takes advantage of information in the request

to assist in the server selection. The term "hashing" is used throughout the present

application generally to CBR and specifically to any form of routing which examines

10   part or all of the content of a request and then routes the request based on the

content. "URL hashing" is one form of hashing which exploits the "locality" of the

request stream by examining the request information and sending requests to a

server that has previously served this request. While this may result in sending the

request to a server that is not the least-loaded server, it may require less overall

15   work for the entire server farm.

The term "URL" stands for "Universal Resource Locator" and is a method

for naming web objects in the Internet. Using a URL, a user of the Internet can

connect to a file on any computer connected to the Internet anywhere in the world.

A typical URL is a string expression conforming to the following convention:

20   protocol://host name/folder or directory on host/name of file or document. For

example, the URL "http://www.ibm.com/products" is parsed as follows. The

"http" stands for the "HyperText Transport Protocol". This directs the browser

(e.g., Internet Explorer or Netscape) to use the http protocol when accessing the document. The "www.ibm.com" is the host name for the IBM main website. As is well-known, each host name is associated with an IP address via the Domain Name System (DNS), which returns an address corresponding to the host name. For

5   example, an IP address associated with www.ibm.com is "0.1.0.7".

The "/products/" means that there is a folder or subdirectory on the IBM website called "Products". Although not shown, within that folder there may be multiple file names, and by adding one of these file names to the URL the computer inputting the URL will be directed to that file.

10   When using URL hashing, the URL is hashed to give it a unique numerical value, which values are assigned to the URL and stored in a table. Each incoming URL is hashed and sent to a particular server and the identification of that server is stored in the table with the hash value; when the hashed value of an incoming URL matches that of a stored hash value in the table, the request is sent to the same

15   server that it was previously sent to.

Figure 1 illustrates a load balancing system 100 in accordance with the prior art. A network 105 of computer work stations 110a, 110b, 110c, and 110d are connected to a network connection 112 (e.g., the Internet) in a known manner. It is understood that although a network 105 of four computer work stations 110a-110d

20   are shown in Figure 1, a single computer work station connected to the Internet or many more computer work stations than the four shown in Figure1 may be utilized to equal effect.

A URL hashing switch 114 (e.g., a hashing switch from the "ServerIron" family of switches manufactured by Foundry Networks) is coupled between the network connection 112 and a server farm 116. In the example shown in Figure 1, the server farm 116 comprises plural servers 118a, 118b, 118c, and 118d. In

5 accordance with this prior art system, when a user of the computer network 105 inputs a URL into a web browser executing on, for example, work station 110a, the URL is transmitted over the Internet in a known manner and is received by URL hashing switch 114. In accordance with this prior art technique, URL hashing switch 114 hashes the URL and stores the URL in a table. Using the system of

10 Figure 1, the URL hashing switch 114 "decides" which server in server farm 116 will handle each incoming URL, based on its hash value. The URL hashing switch 114 may be pre-configured to direct certain hash values to certain servers, or the hash values can be assigned to servers as the requests arrive based on standard SLB techniques.

15 Some known load balancing methods involve placing a front-end processor before the server farm to distribute the load among the back-end servers by keeping track of the load of the back end servers and routing requests to the least-loaded server, while also exploiting the locality of the request stream by routing repeat requests to the same server. Locality-Aware Request Distribution (LARD) is one

20 such system. Other methods have focused on front-end processors that perform level 4 switching (TCP level switching) to balance the load at back-end servers using a round robin technique. These systems may also store load information about

the back-end servers and use this load information to improve upon the round robin

scheduling (which would otherwise not consider the load of the servers). The IBM

Network Dispatcher ™ is one such system implemented in software. Other vendors

implement these types of systems in switches. Level 4 switching techniques do not

5   attempt to take advantage of the locality of the request stream, meaning that requests

that may already have been processed by one server for a client may be sent to a

different server for a different client.

While each of the above methods operate reasonably well, each method

involves sending the request through a routing switch for determining to which

10   server to send the request.

Summary of the Invention

The present invention provides an improved load balancing method, system,

and computer program product in which popular requests are handled by a front-end

cache, and hashing is applied only to the requests in the stream that were not

15   handled by the front-end cache.

In a preferred embodiment, a web proxy cache is placed in front of a Level 7

switch, such that the web proxy cache services the popular requests from the cache

based on the URL, i.e., based on the portion of the HTTP request following the

domain name. The remaining requests are URL-hashed and then routed to the back-

20   end server. This allows the requests that make it past the web proxy cache to still

be routed to the back-end server cache and take advantage of the efficiencies

provided therefrom. Preferably the web proxy cache and the Level 7 hashing switch

are combined as a single functional element, either in hardware or software.

In a more preferred embodiment, a Level 4 switch is placed in front of a

plurality of web proxy caches, each of which are in turn placed in front of (or

5   combined with) a respective Level 7 switch, each of which are connected to a

respective server farm, so that incoming web requests are handled on a round robin

basis before being sent to the web proxy cache, thus improving the throughput from

the server farms to the requesting clients.

Brief Description of the Drawings

Figure 1 is a block diagram of a load balancing system in accordance with the

prior art;

Figure 2 is a block diagram of an improved load balancing system in

accordance with the present invention;

Figure 3 is a block diagram of an example of a Caching/Hashing Switch 214

15   in accordance with the present invention;

Figure 4 is a block diagram of a hierarchical load balancing system in

accordance with the present invention; and

Figure 5 is a flow chart illustrating the basic steps carried out in accordance

with the present invention.

Detailed Description of the Preferred Embodiments

Figure 2 is a block diagram of an improved load balancing system in accordance with the present invention. Referring to Figure 2, a group of clients 205, comprising for example, plural workstations 210a, 210b, 210c, and 210d is

5   connected via network connection 212 (e.g., the Internet), to a "Caching/Hashing Switch" (CHS) 214, which is in turn connected to a server farm 216 comprising, for example, plural web servers 218a, 218b, 218c, and 218d.

In accordance with the present invention, CHS 214 performs both a caching and switching function. It is understood, however, that the same novel functionality

10   can be accomplished using a cache situated in front of a separate hashing switch. When a request is received for the first time by CHS 214, the requested object will not be found in the cache. The term "object" as used herein refers generally to files, HTML responses, or any other content that a client might request from a server. The request is then hashed using known hashing techniques and sent to the

15   destination server. The requested object is delivered to the CHS 214 where it is stored, and a copy of the requested object is sent on to the requesting client.

For subsequent requests for objects that have previously been requested, they will be either served directly from the cache if they are still present (e.g., if the storage period for the cache, relative to the request, has not expired) or they are

20   hashed again and sent to the same server that served them the first time.

Figure 3 is a block diagram of an example of a combined CHS 214 in accordance with the present invention. Referring now to Figure 3, CHS 214

7

comprises a single caching/hashing element, i.e., it is a single functional element having a built-in web caching and URL hashing function. How to build these combined functions into a single functional element will be apparent to those skilled in the art.

5   CHS 214 includes a web proxy cache 330 and a URL-hashing switch 332. The web proxy cache 330 is coupled to the client side via the network connection and is also coupled to the URL-hashing switch 332. URL-hashing switch 332 is in turn connected to the server farm via connection 336. By configuring the CHS 214 to first satisfy any web requests from the web proxy cache 320 (by looking for the

10   requested object in the cache) and, if a "cache-miss" occurs, only then sending the request on to the URL-hashing switch 332 via connection 334, previously requested popular web requests are satisfied without having to burden the server farm, thereby increasing the efficiency of the overall operation of the system. Configuration of these functions can be accomplished by any known methodology; for example, it is

15   well within the ordinary skill of a programmer to be able to write software code to control the CHS 214 so that stored objects for which requests are received are satisfied by return of the cached object, and to hash the request and forward it to the server farm to be satisfied only if the requested object is not found in the cache.

In a preferred embodiment, a hierarchical system is employed as shown in

20   Figure 4. The system illustrated in Fig. 4 will help minimize server farm throughput "bottlenecks" that may occur due to having a proxy cache at the front end of the system. As shown in Figure 4, a Level 4 switch 440 is employed

between the client devices 405/Network Connection 412 and multiple CHS's 414

and server farms 416 as shown. Level 4 switch 440 performs round robin routing

(or any other SLB technique) to the CHS's 414. A benefit of this approach is that

the Level 4 switches can be bought "off the shelf" and combined with the CHS of

5   the present invention (or a functional equivalent thereof) to provide greater

scalability. The system illustrated in Fig. 4 is arbitrarily scalable because more

CHS elements may be added between the Level 4 switch and the server farms (and

thus additional server farms may also be added).

Figure 5 is a flowchart illustrating the basic steps of operation of the present

10   invention. At step 502, a request for a web object is received by the CHS (214 in

Fig. 2) and at step 504, the web proxy cache of the CHS is searched to determine if

the requested object is stored therein (step 506). If the object is already stored in

the web proxy cache, at step 510 the object is immediately returned to the client

directly from the web proxy cache, with no need to burden the server with fulfilling

15   the request, and the process ends at step 516.

On the other hand, if, at step 506 it is determined that the object is not in the

web proxy cache, at step 508 the URL of the object is hashed in a known manner

and, at step 512, the hash value is sent to the destination server. At step 514, upon

receipt of the object request, in the form of the hash value, the server returns the

20   object to the web proxy cache, stores the object in the web proxy cache for future

reference, and then sends the requested object to the requesting client. Finally, at

step 516 the process ends.

The process utilized by the hierarchical system illustrated in Fig. 4 is essentially identical, except that, prior to delivery of the web object request from the client, the requests pass through Level 4 switch 440 which performs conventional round robin routing to the individual cache/switches 414.

5   Using the approach of the present invention, client connections are always terminated in the web proxy cache and the URL hashing may be performed on each HTTP request not found in the web proxy cache for determining which back end server to route the request to. This means that even with HTTP 1.1, it is possible to balance the load at the http request level and to take advantage of the locality of each

10   request. HTTP 1.1 added extensions to the HTTP protocol, one of which allowed multiple requests to flow over a single connection from client to web (or web proxy) server. This invention allows requests within the single connection to be routed independently.

       By handling "popular" web requests using a front-end cache and then

15   utilizing hashing techniques for the un-cached requests, the load-balancing for the remaining requests can be maximized, and the system overall is arbitrarily scalable since, as described above, more CHS's may be added behind the Level 4 switch and more back-end servers may be added behind the additional CHS's.

       Although the present invention has been described with respect to a specific

20   preferred embodiment thereof, various changes and modifications may be suggested to one skilled in the art and it is intended that the present invention encompass such changes and modifications as fall within the scope of the appended claims.